Apache Arrowフォーマットはなぜ速いのか

須藤功平

株式会社クリアコード

爆速DB「PG-Strom」勉強会
ビッグデータをApache Arrowで爆速にしよう
2024-12-16

Apache Arrowと私

- ✓ 2016-12-21に最初のコミット
- ✓ 2017-05-10にコミッター
- ✓ 2017-09-15にPMCメンバー
- ✓ 2022-01-26にPMCチェア
- ✓ 2024-12-16現在コミット数1位

Apache Arrow

- ✓ データ分析ツールの基盤を提供
- ✓ ツールで必要になるやつ全部入り
- ✓ 各種プログラミング言語をサポート

全部ってなに！？

データフォーマットとかそのデータを高速処理する機能とか他の各種データフォーマットと変換する機能とかローカル・リモートにあるデータを透過的に読み書きする機能とか高速RPCとかなんだけど、全部を説明すると「すごそうだけどよくわからないね！」と言われる！

今日のトピック

Apache Arrow フォーマット

Apache Arrowフォーマット

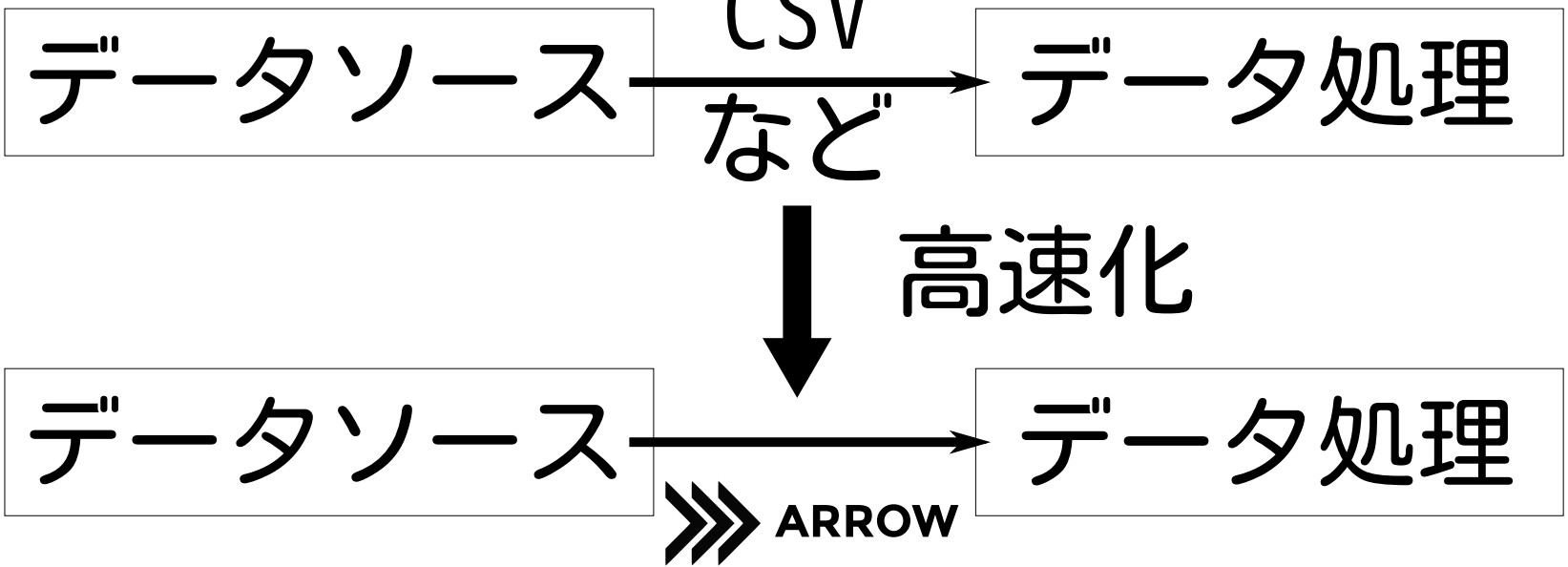
- ✓ データフォーマット
  - ✓ 通信用とインメモリー用の両方
- ✓ 表形式のデータ用
  - ✓ ＝データフレーム形式のデータ用
- ✓ 速い！

速い！

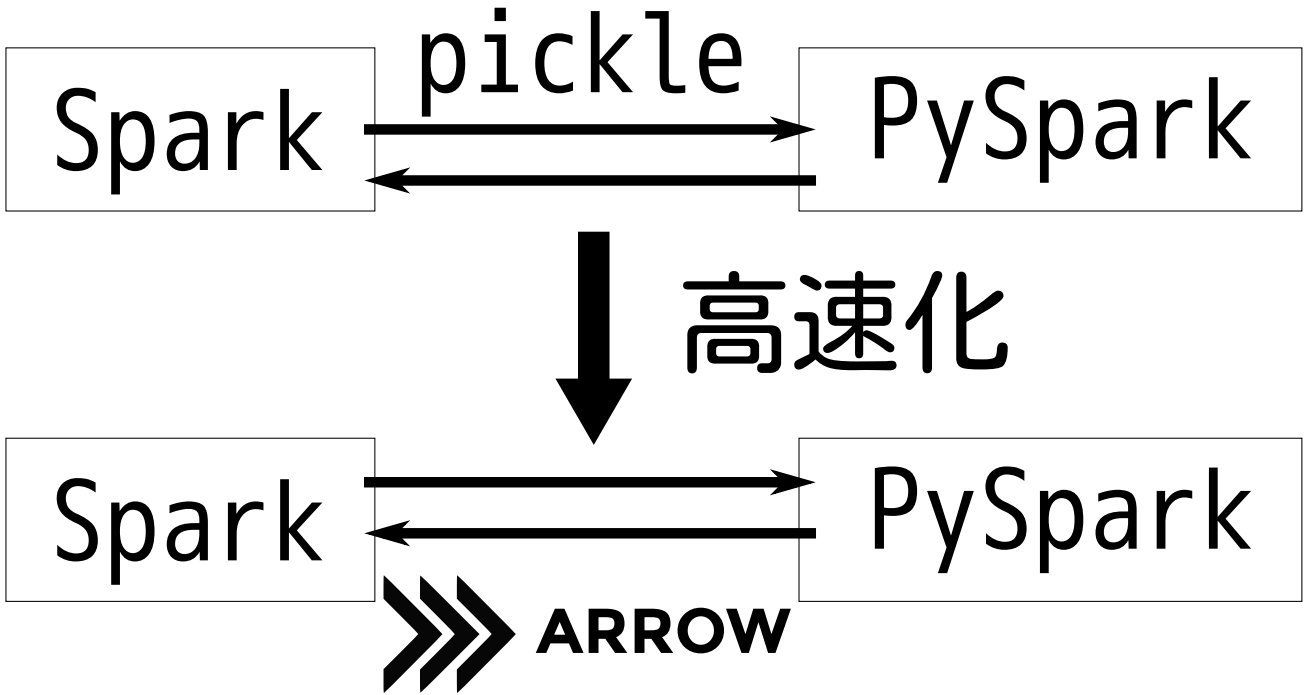
- ✓ データ**交換**が速い！
- ✓ データ**処理**が速い！

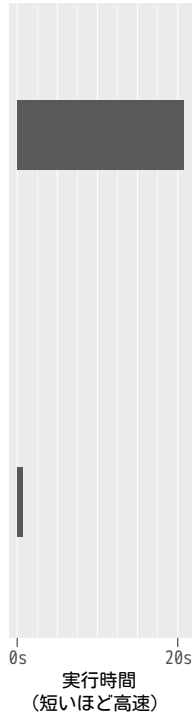
データ交換が速い！

Apache Arrowフォーマットにすると高速化！



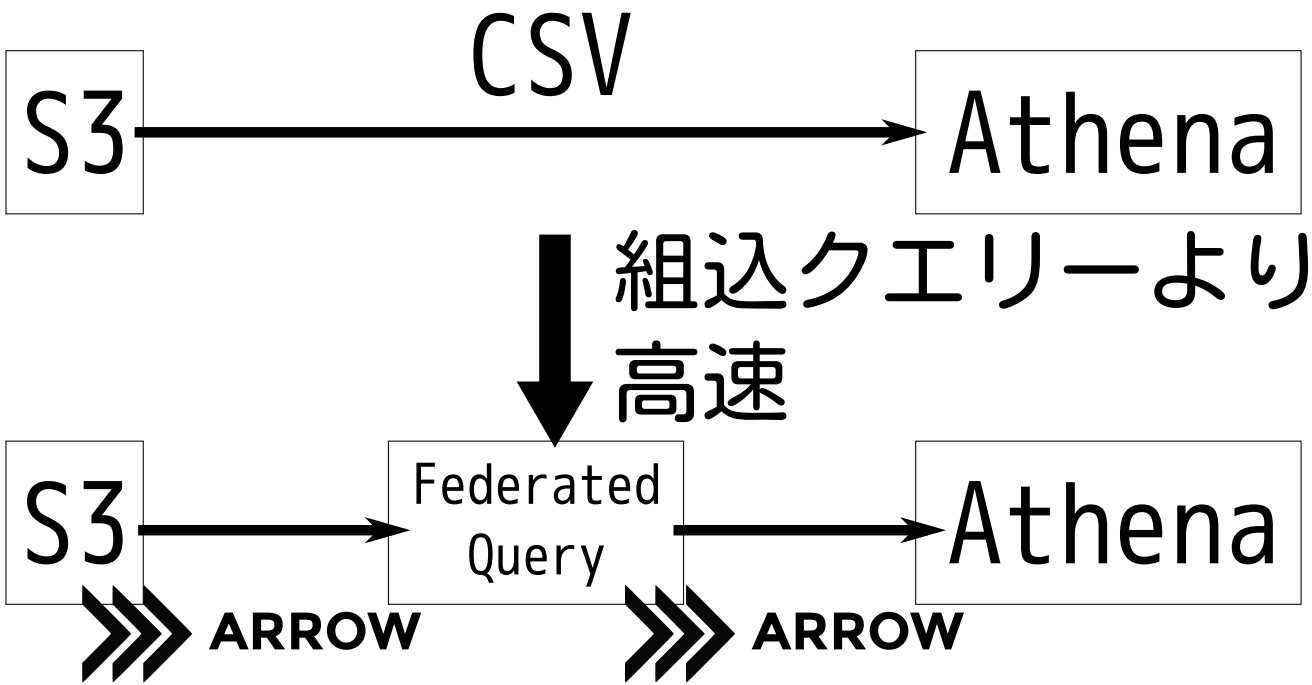
利用事例：Apache Spark

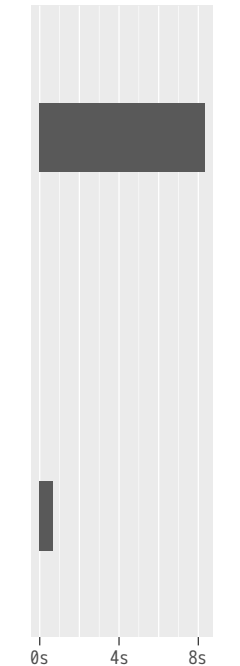




Data at https://arrow.apache.org/blog/2017/07/26/spark-arrow/

利用事例：Amazon Athena





1億行の処理にかかった時間
（短いほど高速）

Data at https://github.com/awslabs/aws-athena-query-federation/tree/master/athena-federation-sdk#performance

利用事例：DuckDB

```
import duckdb
import pyarrow.parquet as pq

lineitem = pq.read_table('lineitemsf1.snappy.parquet')
con = duckdb.connect()
# Transforms Query Result from DuckDB to Arrow Table
con.execute("""SELECT sum(l_extendedprice * l_discount) AS revenue
               FROM lineitem;""").fetch_arrow_table()
```

https://arrow.apache.org/blog/2021/12/03/arrow-duckdb/

利用事例：PG-Strom

```
IMPORT FOREIGN SCHEMA flogdata
  FROM SERVER arrow_fdw
  INTO public
  OPTIONS (file '/path/to/logdata.arrow');
```

https://heterodb.github.io/pg-strom/ja/arrow_fdw/

Apache Parquetだと読み込み処理がボトルネックになっちゃったけど
Apache Arrowだとそれを解消できたんだって

どうして速いの？

- ✓ シリアライズコストが低い
  - ✓ すぐに送れるようになる
- ✓ デシリアライズコストが低い
  - ✓ 受け取ったデータをすぐに使えるようになる

シリアライズ処理

1. メタデータを用意
2. メタデータ+元データそのものを送信
   - ✓ 元データを加工しないから速い！
   - ✓ なにもしないのが最速！

元データを加工する例：JSON

```
0x01 0x02（8bit数値の配列）
    ↓
"[1,2]"（JSON）
0x01→0x49（数値→ASCIIの文字：'1'）
0x02→0x50（数値→ASCIIの文字：'2'）
```

元データそのものを使うと…

- ✓ 変換処理にCPUを使わなくてよい
  - ✓ 速い
- ✓ 変換後のデータ用のメモリー確保ゼロ
  - ✓ 大きなメモリー確保はコストが高い
  - ✓ 一定の作業領域を使い回すとかしなくてよい
  - ✓ 速い

デシリアライズ処理

1. メタデータをパース
2. メタデータを基に元データを取り出す
   - ✓ 元データをそのまま使えるから速い！
   - ✓ なにもしないのが最速！

元データを元に戻す例：JSON

```
"[1,2]" (JSON)
    ↓
0x01 0x02 (8bit数値の配列)
0x49→0x01 (ASCIIの文字：'1'→数値)
0x50→0x02 (ASCIIの文字：'2'→数値)
```

元データを取り出せると…

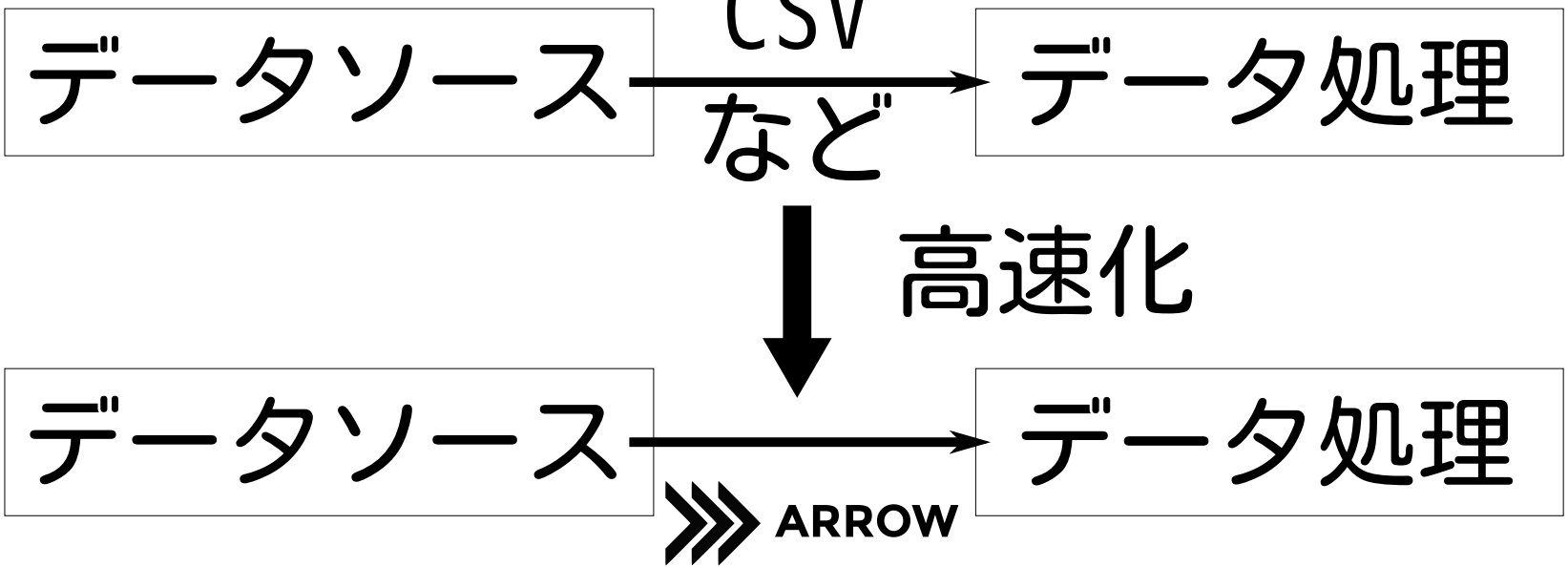
- ✓ 変換処理にCPUを使わなくてよい
  - ✓ 速い
- ✓ 変換後のデータ用のメモリー確保ゼロ
  - ✓ すでにあるデータをそのまま使うのでゼロコピー
  - ✓ 速い
- ✓ メモリーマップで直接データを使える
  - ✓ メモリーより大きいストレージ上のデータを扱える

{,デ}シリアライズコスト

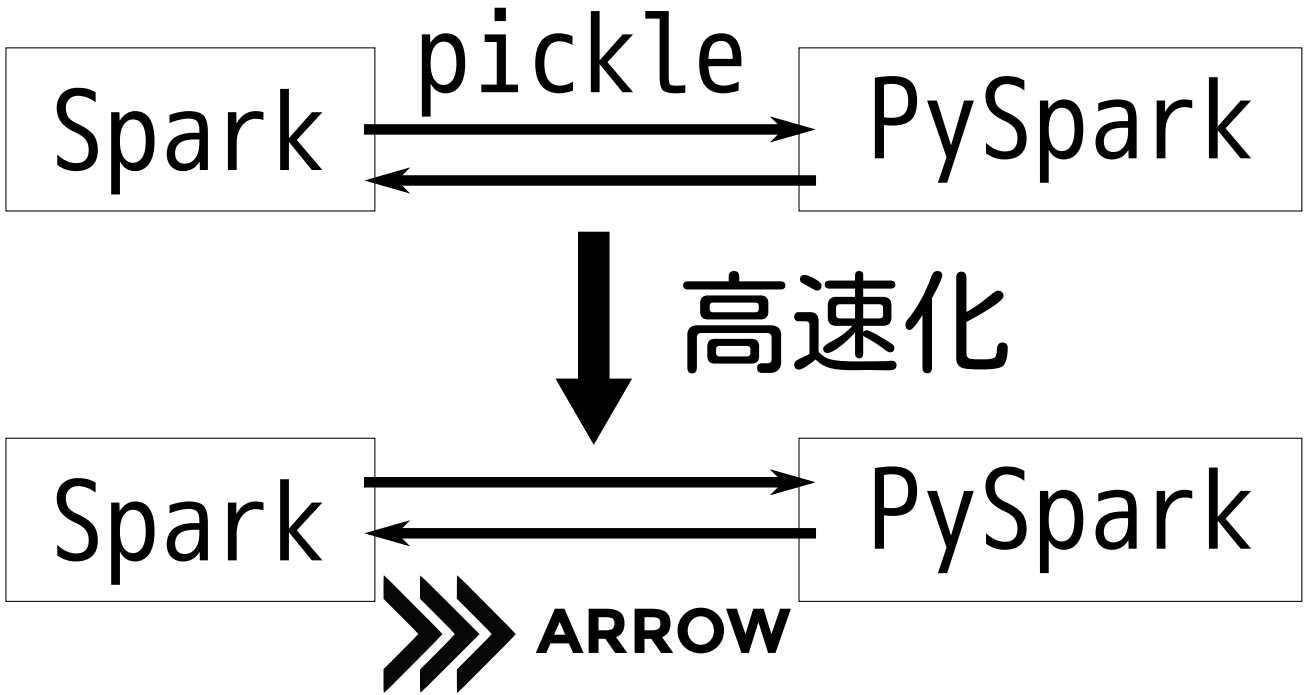
- ✓ Apache Arrowフォーマット
  - ✓ ほぼメタデータのパースコストだけ
- ✓ それ以外の多くのフォーマット
  - ✓ データ変換処理（CPU）
  - ✓ 作業用メモリー確保処理（メモリー）

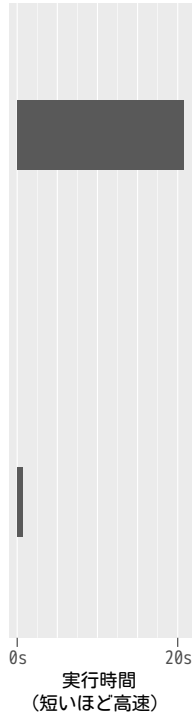
データ交換が速い！

Apache Arrowフォーマットにすると高速化！



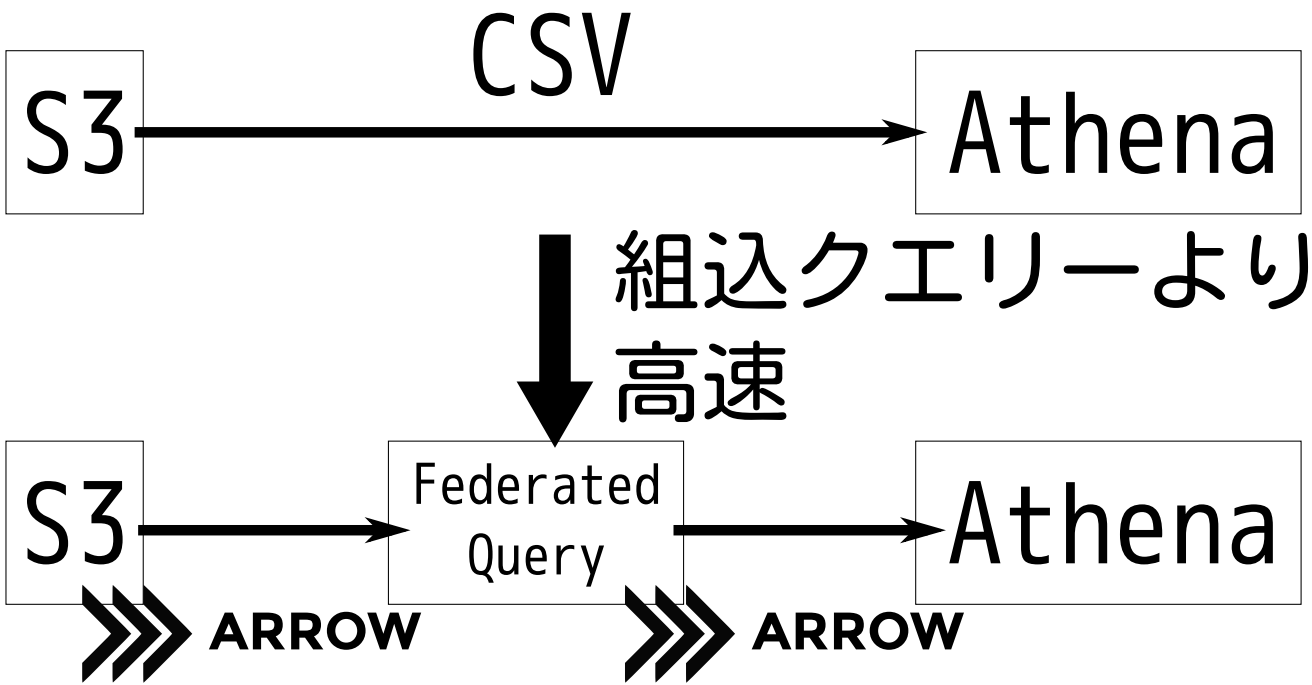
利用事例：Apache Spark

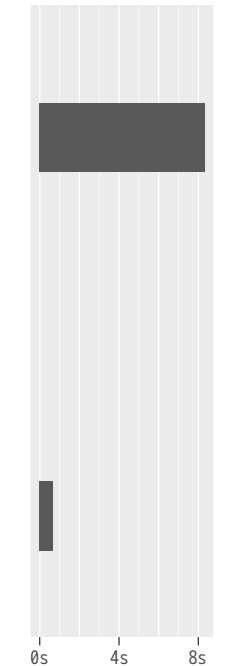




Data at https://arrow.apache.org/blog/2017/07/26/spark-arrow/

利用事例：Amazon Athena





1億行の処理にかかった時間
（短いほど高速）

Data at https://github.com/awslabs/aws-athena-query-federation/tree/master/athena-federation-sdk#performance

利用事例：DuckDB

```
import duckdb
import pyarrow.parquet as pq

lineitem = pq.read_table('lineitemsf1.snappy.parquet')
con = duckdb.connect()
# Transforms Query Result from DuckDB to Arrow Table
con.execute("""SELECT sum(l_extendedprice * l_discount) AS revenue
               FROM lineitem;""").fetch_arrow_table()
```

https://arrow.apache.org/blog/2021/12/03/arrow-duckdb/

利用事例：PG-Strom

```
IMPORT FOREIGN SCHEMA flogdata
  FROM SERVER arrow_fdw
  INTO public
  OPTIONS (file '/path/to/logdata.arrow');
```

https://heterodb.github.io/pg-strom/ja/arrow_fdw/

Apache Parquetだと読み込み処理がボトルネックになっちゃったけど
Apache Arrowだとそれを解消できたんだって

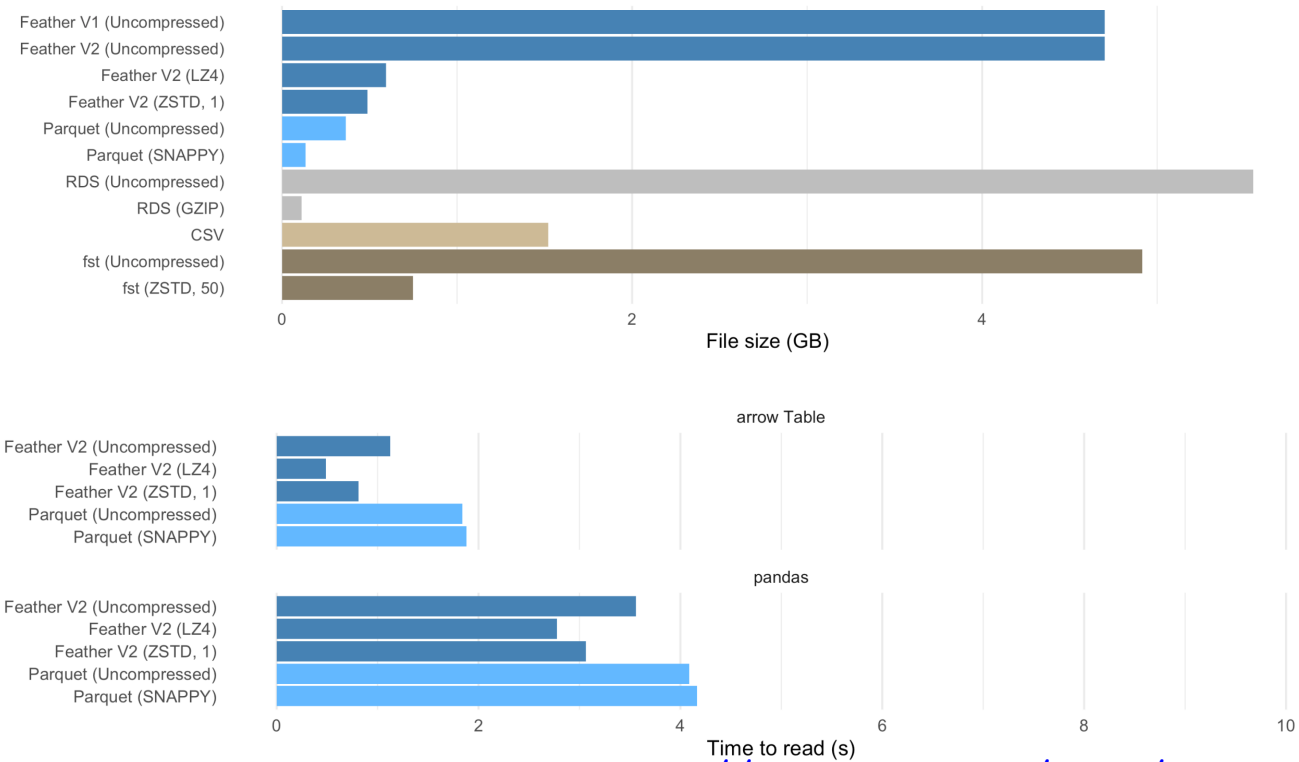
データサイズは？

- ✓ CPU・メモリーにやさしくて速いんだね！
- ✓ じゃあ、データサイズはどうなの？
  - ✓ 大きいとネットワーク・IOがボトルネック
  - ✓ 本来のデータ処理に最大限リソースを使いたい
  - ✓ データ交換をボトルネックにしたくない

データサイズ

- ✓ 別に小さくない
  - ✓ 元データそのままなのでデータ量に応じて増加
- ✓ Zstandard・LZ4での圧縮をサポート
  - ✓ ゼロコピーではなくなるがサイズは数分の1
  - ✓ 圧縮・展開が必要→元データそのものを使えない
  - ✓ CPU・メモリー負荷は上がるが
    ネットワーク・IO負荷は下がる
  - ✓ ネットワーク・IOがボトルネックになるなら効く

圧縮時のデータサイズと読み込み速度



https://ursalabs.org/blog/2020-feather-v2/

データ交換が速い！のまとめ

- ✓ {,デ}シリアライズが速い
  - ✓ 元データをそのまま使うので処理が少ない
  - ✓ CPU・メモリーにやさしい
- ✓ 圧縮もサポート
  - ✓ ネットワーク・IOがボトルネックならこれ
  - ✓ CPU・メモリー負荷は上がるが
    データ交換のボトルネックを解消できるかも

交換したデータの扱い

- ✓ データ分析はデータ交換だけじゃない
  - ✓ データ交換だけ速くしても基盤とは言えない
- ✓ データ処理も速くしないと！
  - ✓ データ処理を速くするにはデータ構造が大事

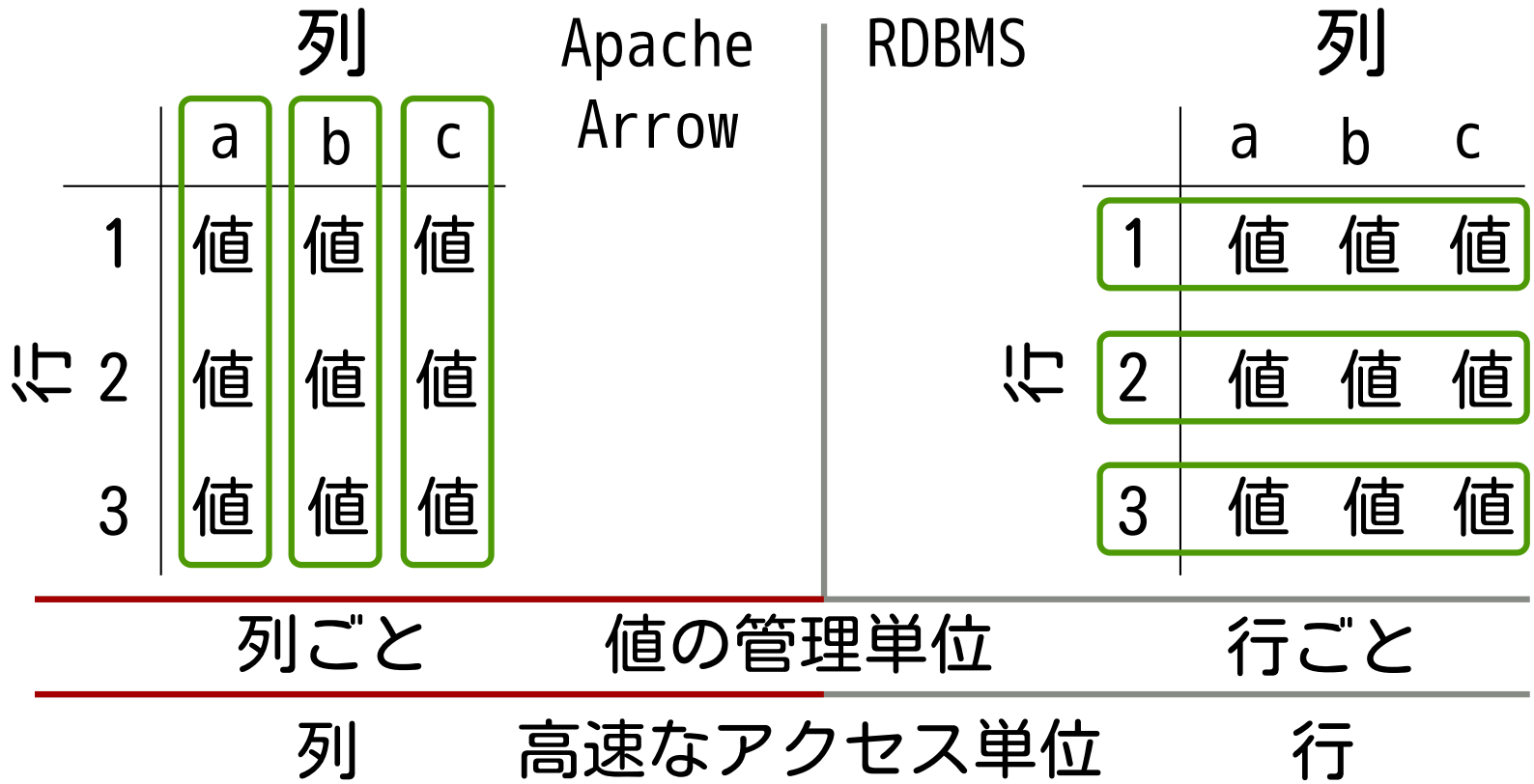
高速処理のためのデータ構造

- 基本方針：
  - 関連するデータを近くに置く
- 効果：
  - CPUキャッシュミスを減らす
  - SIMDを活用できる

データ分析時の関連データ

- ✓ 分析時はカラムごとの処理が多い
  - ✓ 集計・ソート・絞り込み…
- ✓ 同じカラムのデータを近くに置く
  - ✓ カラムナーフォーマット

カラムナーフォーマット



各カラムでのデータの配置

- ✓ 関連するデータを近くに置く
- ✓ 定数時間でアクセスできるように置く
- ✓ SIMDできるように置く
  - ✓ アラインする
    アライン：データの境界を64の倍数とかに揃える
  - ✓ 条件分岐をなくす

真偽値・数値のデータの配置

固定長データなので連続して配置
32ビット整数：[1, 2, 3]
0x01 0x00 0x00 0x00 0x02 0x00 0x00 0x00 0x03 ...

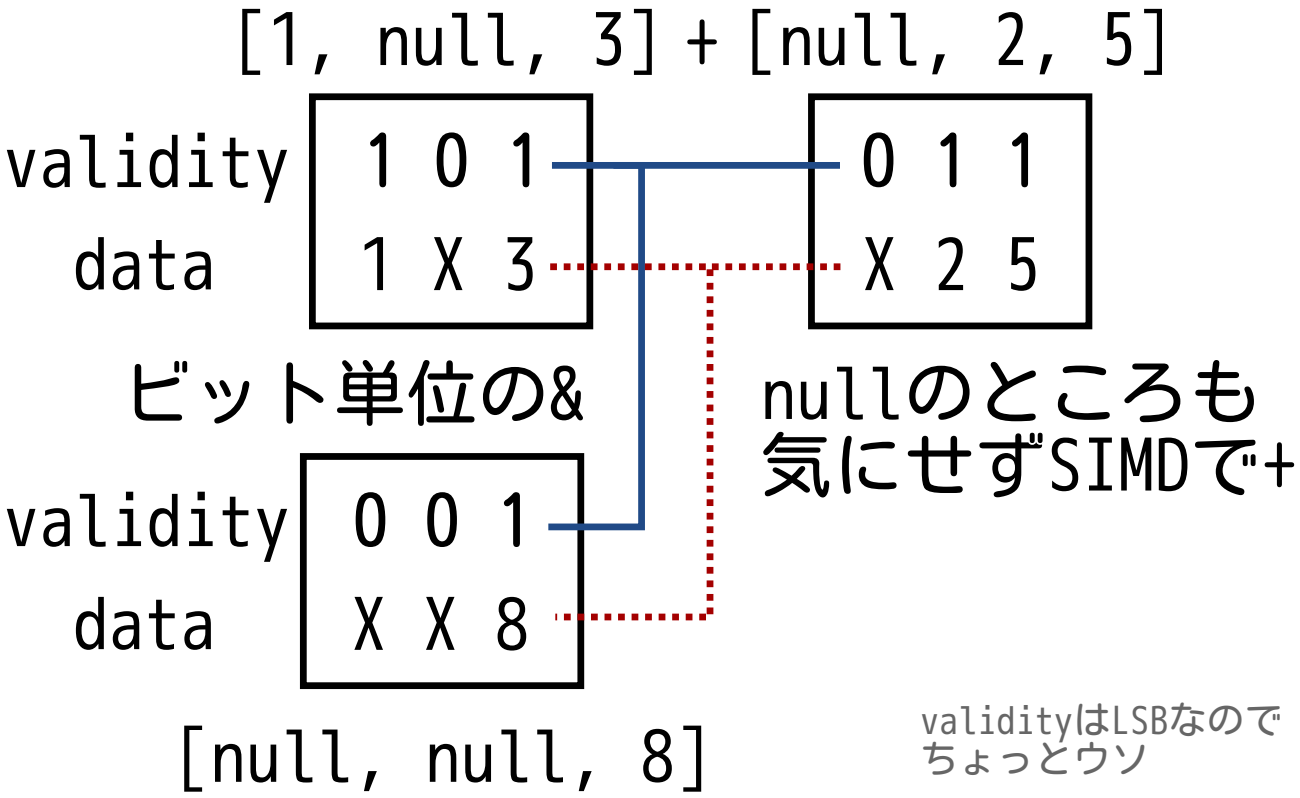
文字列・バイト列：データの配置

実データバイト列+長さ配列に配置
UTF-8文字列：["Hello", "", "!"]
実データバイト列："Hello!"
長さ配列：[0, 5, 5, 6]
i番目の長さ：長さ配列[i+1] - 長さ配列[i]
i番目のデータ：
　実データバイト列[長さ配列[i]..長さ配列[i+1]]
注：長さ→データ→長さ→データ→…で置くと
　　定数時間でi番目にアクセスできない

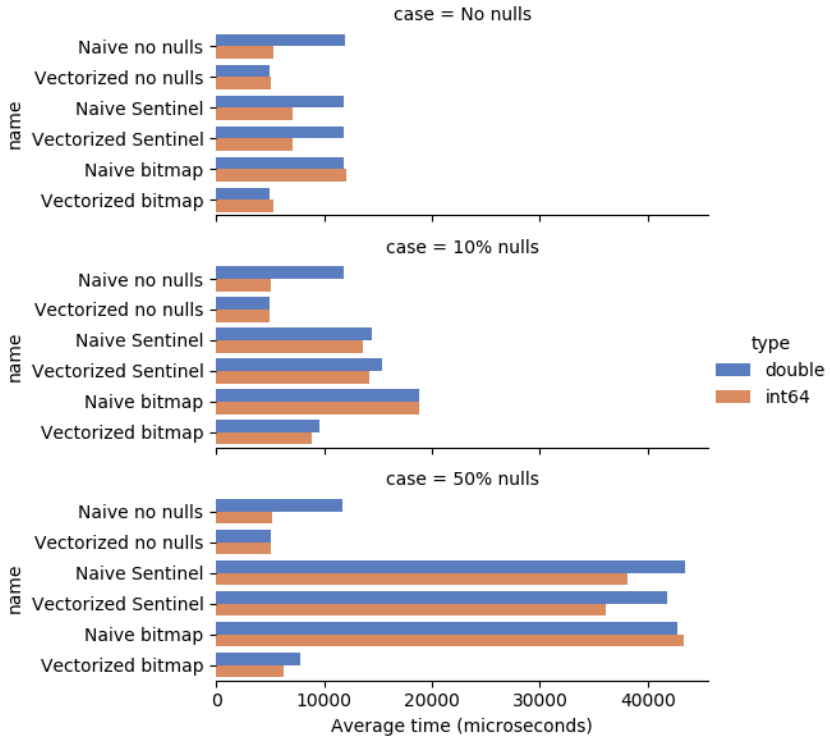
nullと条件分岐

- ✓ null対応のアプローチ：
  - ✓ null値：Julia: missing, R: NA
  - ✓ 別途ビットマップを用意：Apache Arrow
- ✓ ビットマップを使うと条件分岐をなくせる

nullと条件分岐とSIMD



nullと条件分岐とSIMD



https://wesmckinney.com/blog/bitmaps-vs-sentinel-values/

高速なデータ処理のまとめ

- ✓ 高速なデータ処理にはデータ構造が重要
  - ✓ データ分析にはカラムナーフォマットが適切
- ✓ 定数時間でアクセス可能なデータの配置
- ✓ SIMDにやさしいデータの持ち方
  - ✓ アライン・null用のビットマップ

まとめ

- ✓ Apache Arrow
  - ✓ なんかデータ分析に便利そうなすごいやつ！
  - ✓ 「よくわからん」と言われるからといって全体像を説明してもらえなかった！
  - ✓ 使い方も説明してもらえなかった！
- ✓ Apache Arrowフォーマット
  - ✓ これだけ説明してもらえた！

まとめ：Apache Arrowフォーマット

- ✓ Apache Arrowフォーマット
  - ✓ 通信用・インメモリー用のデータフォーマット
  - ✓ 表形式のデータ用
- ✓ Apache Arrowフォーマットは速い
  - ✓ データ**交換**が速い
  - ✓ データ**処理**が速い
  - ✓ データ交換してすぐに高速処理できるフォーマット

まとめ：なぜデータ交換が速いのか

- ✓ 元データをそのままやりとりできるから
  - ✓ {,デ}シリアライズコストが低い
  - ✓ CPU・メモリーにやさしい
- ✓ ネットワーク・IOの負荷を下げたい
  - ✓ Zstandard・LZ4による圧縮
  - ✓ CPU・メモリー負荷は上がるが
ネットワーク・IO負荷は下がる

まとめ：なぜデータ処理が速いのか

- ✓ 最適化されたデータ構造
  - ✓ カラムナーフォーマット
  - ✓ SIMDを使えるデータの持ち方
- ✓ 最適化された実装
  - ✓ 今回は紹介していない！！！

おまけ

他のApache Arrow関連情報を
ざっくり紹介

Apache Arrow Flight

- ✓ Apache Arrow + gRPCの高速RPC
  - ✓ gRPC→UCXもできたけどやめた
  - ✓ UCXを使いたいときはDissociated IPC Protocol
    https://arrow.apache.org/docs/format/DissociatedIPC.html
- ✓ SQL向け拡張Flight SQLもある
  - ✓ PostgreSQL向け実装を作ってみた
    https://github.com/apache/arrow-flight-sql-postgresql

ADBC

- ODBCのApache Arrow版みたいなやつ
  - クライアント側で動く
- 入力・出力データがApache Arrow
  - サーバー側がApache Arrow非対応でも
    クライアント側（ADBC内）で変換
  - アプリ側でもできるけど
    みんなでADBCに高速な実装を入れて
    共有しようぜ！というアプローチ

ADBC + PostgreSQL

- ✓ PostgreSQLアダプターはpgeonインスパイア
- ✓ pgeonはpg2arrowインスパイア
- ✓ pg2arrowはPG-Stromに同梱
- ✓ 高速化例：SELECT結果の送信にCOPYを使う

https://www.clear-code.com/blog/2024/12/6/postgresql-conference-japan-2024.html

PostgreSQL + Apache Arrow

- ✓ COPYでApache Arrowを使えたらいいね
- ✓ 今のCOPY：text/csv/binaryのみ
- ✓ COPYで扱えるフォーマットを拡張したい！

  pgsql-hackers: Make COPY format extendable: Extract COPY TO format implementations
  https://www.postgresql.org/message-id/flat/20231204.153548.2126325458835528809.kou@clear-code.com

ファイルシステムの抽象化

- ✓ ローカル
- ✓ HDFS
- ✓ オブジェクトストレージ
  - ✓ S3/GCS/Azure Blob Storage

データセット

- ✓ 複数データを論理的に1つとして扱う
- ✓ データ置き場：
  - ✓ 抽象化されたファイルシステム
- ✓ データフォーマット：
  - ✓ Apache Arrow, Apache Parquet, CSV, JSON, ...

クエリーエンジン：Acero

- ✓ データセットをストリーム処理
  https://arrow.apache.org/docs/cpp/streaming_execution.html
- ✓ 結構動いているんだけど
  今後どうなるかはわからない